Router Solutions using Web Services Gateway

This chapter provides information on using the Web Services Gateway to implement an Application Integration solution in both an intra-enterprise and inter-enterprise environment.

It includes information on:

- The Web Services Gateway capabilities
- Applying patterns
- Runtime patterns and product mappings
- System design and design guidelines
- Development guidelines and example
- Quality of Service considerations
**Business scenario**

In this scenario, ABC Electronics consists of multiple Retail stores and Wholesale departments. The Retail stores get their supplies from the Wholesale departments and have a need to request the delivery dates of those supplies.

Currently there is no integration of the Retail and Wholesale applications. All interactions between the two are done over the phone or by mail. In this scenario, a solution must be found to allow the Retail stores to request delivery dates from the Wholesale departments.

The solution requires the following elements:

- A central access point for all services inside the enterprise.
- Decoupling Retail store functions from the Wholesale department functions so that changes made to the Wholesale department systems or their location do not require a change to the Retail store systems.
- A central security control point that controls access to the Wholesale department services.
- Protocol conversion between the Retail store systems and the Wholesale department service providers to accommodate the addition of new services. The transition of protocols used between the Retail and Wholesale systems must be transparent.

ABC Electronics is planning to implement the solution in three stages:

- **Stage 1: Basic integration:**
  The Retail system requests either the service of Wholesale1 or the service of Wholesale2, therefore the Retail system has to know which Wholesale holds the part it needs.

- **Stage 2: Enhanced integration:**
  ABC Electronics enhances this solution, so that the Retail system no longer needs to know where the part is.

- **Stage 3: Extended enterprise:**
  ABC Electronics wants to allow external Retail systems to get delivery dates from the Wholesale departments and to plan for allowing internal Retail systems to get delivery dates from external Wholesale suppliers.
IBM Web Services Gateway

The IBM Web Services Gateway is a runtime component that provides configurable mapping between Web service providers and requesters. Services defined with WSDL can be mapped to available transport channels. The Web Services Gateway is included with the IBM WebSphere Application Server Network Deployment and Enterprise packages.

These are the basic gateway components:

- Channels that define the entry-points into the gateway and carry the Web service request and response through the gateway.
- Filters that are used to intercept service invocations which come into the gateway and act upon the services.
- Services that are described with the help of a Web Services Description Language (WSDL) document.
- UDDI references to manage the publishing of an exposed Web service to a private or public UDDI registry.

Figure 1 shows the relationship between the first three components. The entry point to the gateway is defined by a channel. A channel is a piece of software that defines the protocol you can use to access the gateway. The incoming message is assessed on arrival through the channel to determine which service is required. Each service (defined in a WSDL document) has to be bound to one or more channels. One or more filters can be bound to a service for manipulating both request and response messages. The WSDL service definition specifies the provider service interface and implementation used to access the target service.

![IBM Web Services Gateway Diagram](image-url)
A request to the Web Services Gateway arrives through a channel and is translated into an internal representation of the service. With the help of filters for the request, a request can be logged, intercepted, or generally manipulated. After filtering the request, an appropriate provider is used to communicate with the target service. The provider in the gateway acts as a client for the target Web service.

The response from the target service flows along the exact same path back to the provider. There is no extra channel for an immediate response. In this sense the layout of the gateway is asymmetric. However, one or more response filters can be deployed independently of the request filters.

The process of deploying a target service into a gateway channel generates two different external WSDL files; an implementation definition and an interface definition. These new WSDL files can be exported for use by client applications, and are the externalization of the service capabilities offered by the internal target service. The implementation WSDL definition is used to simplify the connection process for a client, particularly when dynamic invocation is being used. Having obtained the implementation definition, the client can then access the WSDL interface definition produced by gateway, which provides full information about the target service (as presented externally by the gateway).

The Web Services Gateway uses the Web Services Invocation Framework (WSIF) API from Apache to decouple invocation from deployment within the gateway. Over time, the location of the Web service target application and the bindings may change, but these details are handled by the gateway. The Web Services Gateway separates the actual implementation of a service from how it is accessed by another service for:

- **Inbound requests**: To Web services created and deployed within the organization.
- **Outbound requests**: To Web services created and deployed outside the organization.
- **Process abstraction**: The service invocation approach must be flexible enough to cope with events such as switching frequently between external providers of a similar service without requiring changes to the application.
- **Flexibility**: As a service provider, you need the flexibility to change your deployment infrastructure without notifying all the service requestors. For example, consider a Web service deployed in a machine that later fails during operation. There needs to be a process to route the invocations to an alternate service in your infrastructure.

WSIF is used within Web Services Gateway as shown in Figure 2. It demonstrates the WSIF transformation from a SOAP message to a target service:
1. The SOAP message arrives at the gateway and the channel listener accepts the message.

2. The channel converts the SOAP message into a WSIF message format.

3. Elements within the message are used to locate the appropriate target service, which is bound to the channel within the gateway.

4. The target WSDL associated with the gateway service is then processed by WSIF.

5. WSIF dynamically generates a Java proxy class.

6. The target Web service is called.

Refer to the following IBM developerWorks articles for further details:

- *Applying the Web services invocation framework*
  

- *An introduction to Web Services Gateway*
  

- *Business process integration with IBM CrossWorlds, Part 3: Automatically externalize Web services with WebSphere Business Connection*
  
Applying patterns

The Retail system needs to get delivery dates based on a part or system number from the appropriate Wholesale supplier. Each Wholesale supplier has its own unique implementation that returns the delivery date to requesters but has basically the same input and output formats for incoming requests and outgoing responses. Each Wholesale system does not necessarily support the same protocol for transmission of the requests and responses.

From this, ABC determined the following requirements:

- Each Wholesale department must expose its services as Web services.
  The actual implementation on the Wholesale systems must be transparent to the Retail system. The request for a delivery date will be done as a Web service invocation, relieving the Retail system from having to know how the request is fulfilled.

- The design must be able to support multiple protocols and perform protocol transformation.
  For the same Web service, Wholesale1 offers a SOAP/HTTP implementation, while Wholesale2 offers a SOAP/JMS implementation. Although the Retail system will initially use SOAP/HTTP as an interface, future requirements will include the need to access the services using SOAP/JMS.

Application patterns

A Router node provides the multi-protocol transformation and intelligent request routing required for the solution, therefore ABC Electronics decided to use the Application Integration Router pattern to design the solution.

To phase in the solution, they decided to implement it in stages.

Stage 1: Basic integration

In the first stage, the Retail system requests either the service of the Wholesale1 or Wholesale2 department. This assumes that the Retail system knows which Wholesale department is the supplier for a particular part number. This solution requires simple router functions, which will be supplied by Web Services Gateway.

The request is directed to the Router node, which invokes the Web service from the requested Wholesale department. The Router presents a unified interface for the Retail system and transforms the message to the appropriate transport protocol for the target. It also performs logging of the event.

The solution implemented in this stage by ABC Electronics is shown in Figure 3.
Stage 2: Enhanced integration
In the second stage, ABC Electronics will enhance the solution so the Retail system no longer needs to know which Wholesale department supplies the part. The Retail system can simply send the request to the Router, which will determine the appropriate Wholesale system and invoke the Web service. This places a requirement on the Router that it provide content-based routing.

The solution implemented in this stage by ABC Electronics is shown in Figure 4.

Stage 3: Extended enterprise
In the third stage, ABC Electronics will enhance the solution so that external Retail systems can request delivery dates from the Wholesale departments. The implementation does not change, but the Web Services Gateway is opened up to the externals and protected with a firewall; and additional security concerns are addressed.

The solution implemented in this stage by ABC Electronics is shown in Figure 5.
Similarly, the solution could provide the Retail systems in ABC Electronics could access external Wholesale companies.

In either case, the Router node can be placed on the secure internal network behind the DMZ. The advantage of this configuration is that the gateway can provide access to services using protocols that are not as firewall-friendly as HTTP. For example, the Router node can enable Web service clients to access EJB, JMS, and JavaBean applications.

The Router node could also be located in the DMZ. All the firewalls in our example are configured for network address translation, so SOAP over HTTP can be used to pass through the firewalls. This may not work if the gateway is using RMI to access an EJB on the secure network, for example.
Runtime Patterns and Product mappings

The Product mapping chosen by ABC Electronics is shown in Figure 7, and is valid for intra-enterprise environments.

![Diagram of Product mapping for Router (using WebSphere Web Services Gateway)](#)

**Note:** The 5.0.2.1 fix level is required for SOAP/JMS.

The only addition to the product mapping for extended enterprise is the firewall system. However, note that even in intra-enterprise environments, a firewall system has added benefit, so you may want to consider the following product mappings for both intra and inter-enterprise.

There are actually two variations to consider. One is that the ABC Electronics wants to get delivery dates from external Wholesale suppliers. Figure 8 shows the Runtime pattern and product mapping that makes external Wholesale services available to the ABC Retail systems.
The second possibility is that external Retail systems want to get delivery dates from ABC Wholesale departments. Figure 9 shows the Runtime pattern and product mapping that exposes the ABC Wholesale services to external Retail systems.
Why use the Web Services Gateway?

In a solution that integrates multiple independent applications, key concerns include interoperability, compatibility, and security. While this is important in an intranet environment, it is even more important in an extended enterprise environment because of the diversity of applications, environments, and users who will be interacting with the exposed service.

The Web Services Gateway was selected for this solution for the following reasons:

- The gateway provides a common access point for internal client and external partner applications needing access to internal services. The gateway intercepts and routes requests to the applications providing the service. A change in the location of the service is handled once at the gateway, versus at each potential service requester site.
- The gateway provides a single point for controlling access to external services and hides changes to the external Web services from your internal client applications.

- The gateway secures your Web services, supporting authentication and authorization of Web service requesters. In an inter-enterprise environment, it can act as a reverse proxy, providing indirect access to your internal Web services.

- Filters can be used to validate or manipulate messages traveling to and from the Web service. In this case, a filter is used to determine the proper Web service to use based on the request. There are some built-in filters supplied with the Web Services Gateway, but you can also create your own.

- Requesters using one protocol can invoke Web services from back-end systems using a different protocol. For example, the Web Services Gateway provides the protocol transformation required to allow an HTTP/SOAP client to access a JMS/SOAP service.

**Important:** The Web Services Gateway is supplied with the Network Deployment package in both IBM WebSphere Application Server Network Deployment and IBM WebSphere Application Server - Enterprise. However, the underlying application server makes the difference in whether certain features are supported or not. In order to support the full functionality of the Web Services Gateway, including filter programming, you will need the IBM WebSphere Application Server - Enterprise product.

### Why use a UDDI registry?

A UDDI registry is used by the Router node to obtain the interface description and the implementation description of the Web services provided by the Wholesale systems.

Although it may be easier in the short term to simply put the WSDL files of the Web services on a Web site than to implement a solution using a public or a private registry, consider the following trade-offs:

- Web sites don’t have a discovery protocol that allows an application to search for and download WSDL files.

- As the use of Web services becomes more popular, consumers are increasingly likely to use a UDDI registry to find Web services.

- UDDI registries allow a fine degree of classification for Web services that allows consumers to quickly find the Web services to fit their needs.
Design guidelines

As the solution was being designed, the following topics were considered.

**SOAP messaging style**

SOAP provides support for both RPC style Web services and document-style Web services. In RPC style, Web services are invoked using remote method calls. In document-style messaging, Web Services are invoked by sending a complete well-formed document describing the task to be performed, and possibly, some parametric data.

These are some advantages of SOAP RPC-based Web services:

- They are simpler to develop than SOAP message-oriented messaging.
- They use a higher level API.
- They support strong typing: all calls must conform to the method signature.

The disadvantages of RPC-based SOAP messaging are:

- Higher coupling between the Web service requester and provider because the requester to provider binding includes both methods and parameters.

On the other hand, when using document-style Web services, the XML document is arbitrary. This style is more flexible and can be used to send one-way messages without needing a response (unlike RPC-style) but involves more work for the developer.

Prior to the Web Services Gateway provided with IBM WebSphere Application Server V5.0.2, RPC-style SOAP messages were required. With V5.0.2, support for document-style messages was added. If you are using the Apache SOAP Channel, then the SOAP message format must be RPC style. To handle Document style SOAP messages, use the SOAP/HTTP channel (which supports both RPC style and Document style SOAP messages). We used the RPC-based mechanism in our sample solution.

**SOAP encoding style**

In addition to the messaging style, the SOAP encoding style should be considered. Message parts can be encoded using encoding rules (“encoded”), or may reference a concrete schema definition (“literal”).

WS-I has stated a preference for the use of literal, non-encoded XML.
SOAP caching
Using WebSphere Application Server dynamic caching to cache SOAP messages can significantly reduce the overhead of Web services. A Web service is a candidate for caching if its requests can be grouped into classes based on identifiers, and if the Web service's responses do not contain arbitrarily changing information based on the system state or time.

Caching a SOAP message is not as easy as putting it in a HashMap. You need to determine what makes two SOAP request identical. Only then can a caching framework determine when to send a client a cached response instead of letting the request propagate further into the system. To decide if two SOAP request are identical, you could have to look at the SOAP Body, or at the SOAP Body and some combination of SOAP header values.

Message optimization
The performance of an application based on Web services is affected by three factors:

- Network transmission time.
- The time it takes to handle the messages, including XML parsing, flow management, invocation of the service and the encoding of the final response
- The time the service itself takes to execute

The two first factors can be improved by optimizing the number, size, and structure of the messages you use. For example:

- Don’t pass parameters that you don’t need.
- Don’t use two or more invocations when you can do the same operation implementing a more complex service. A good example for this could be: Instead of repeatedly using an operation that returns the price of a product, implement a service that returns the price for a list of products.

The third factor can be improved using a good caching system, as discussed previously.

Security
Security is certainly important in an intra-enterprise environment, but becomes crucial in an inter-enterprise environment. General security guidelines are:

- Use https when invoking external services.

  The Web Services Gateway can invoke Web services that include https:// in their addresses, if the Java and WebSphere security properties have been configured to allow it.
Protect your services from non-authorized access.

Web Services Gateway provides a basic authentication and authorization mechanism based upon the broader security features of WebSphere Application Server. For more information, see “Basic authentication and authorization facilities” on page 63.

You can also use the WS-Security support of the Web Services Gateway to implement authentication and authorization controls.

When possible, implement security solutions based on common specifications, like WS-Security, facilitating interoperability.

You can configure the gateway for secure transmission of SOAP messages using tokens, keys, signatures and encryption in accordance with the Web Services Security (WS-Security) draft recommendation.

Security issues are discussed more in “Security” on page 63.

SOAP transport protocol

In principle, SOAP is transport protocol independent and can therefore potentially use a variety of protocols (such as HTTP, JMS, SMTP, and others) to connect the Web service requestor and the provider.

HTTP is currently the most popular transport protocol for SOAP. The reasons for this mainly result from the following advantages:

► HTTP is the de facto standard on the Internet
► HTTP is wide spread
► HTTP is supported from most programming languages
► HTTP has a simple extension for security, HTTPS
► HTTP needs no complex infrastructure

But there are also some important limitations:

► HTTP is optimized for use in browser and end-user scenarios
► HTTP is a stateless communication protocol
► HTTP does not provide reliable communication
► HTTP establishes a point-to-point connection

These limitations are usually acceptable for human-to-machine communication using a Web browser. This may not be valid when switching to machine-to-machine communication. The requirements in machine-to-machine communication are usually more complex and other transport protocols may be more suitable.

For example, if you need assured, once-only delivery, message-oriented middleware may be more suitable. In Java you can use Java Messaging Service (JMS) with IBM WebSphere MQ. The advantages are:
- Reliable messaging
- Option of asynchronous communication

On the other hand, you have the following disadvantages:
- More complex infrastructure
- Not generally available for all potential customers
- Message-oriented middleware skills are necessary
- At this time most vendors only have limited support for SOAP over JMS

With the increasing adoption of Web services, the portfolio of supported transport protocols will also increase.

We use the HTTP transport protocol in our sample solution because HTTP can provide the connection we need between source application and target application without the need for added infrastructure.

In addition, HTTP is easier to deal with in an inter-enterprise situation. It is common to allow the HTTP protocol through firewalls and simplifies connectivity to external partners.

**Apache SOAP versus SOAP/HTTP channels**

The Apache SOAP channel and the SOAP/HTTP channel both support SOAP applications that are SOAP 1.1 compatible (for example Apache SOAP 2.3 and Axis SOAP 1.0). So if you have an application that uses a production-supported Axis 1.0 SOAP stack, generating SOAP 1.1, then it can use either channel.

If you are using the Apache SOAP Channel, then the SOAP message format must be RPC style. To handle document style SOAP messages, you must use the SOAP/HTTP channel (which supports both RPC and document style SOAP messages).

If you deploy Web services that pass attachments in a MIME message, then these Web services can only be accessed using the SOAP/HTTP channel.

**Use of filters**

Filters are session beans than intercept requests that arrive at the Web Services Gateway or responses than leave it. Filters can perform almost any task, and they have access to the context and to the message that is intercepted. They run in a separate enterprise application, so when building a filter, you have to consider the following Quality of Service issues:
- Performance
- Availability
- Security
- Standard compliance
For example, if you use the clustering feature of WebSphere Application Server for to enhance the Web Services Gateway performance and availability, take similar measures for the filter application.

Don’t create filters that do a lot of tasks. Instead, create multiple filters that each perform one simple task (for example, logging, transforming content, routing, etc.). Then, associate all the filters to the service you need. This facilitates the development, reduces the complexity of the filters, and promotes the reuse.

System design overview

After considering the design guidelines, ABC Electronics decided to implement and test the solution in the following three stages.

Stage 1: Basic integration

In the first stage, the Retail and Wholesale departments are all internal to ABC Electronics. The Retail system is aware of which Wholesale department is supplying each part and codes the request appropriately. The Router provides transport protocol conversion and simple logging. It also provides a layer of abstraction between the systems that allows the Wholesale services to change or move without affecting the Retail systems.

The solution using WebSphere Web Services Gateway in this stage is shown in Figure 10.

![Figure 10 Web Services Gateway implementation, stage 1](image-url)
1. The client application (Retail) connects to the gateway using SOAP/HTTP to send the request to the channel.

2. The service within the gateway receives the request and resends it to the target service. In this phase, we are using one target service for each Web service defined.

3. The location of the service provider (for example, Wholesale1) is obtained from the WSDL associated with the service. The location of the WSDL can be specified in two ways:
   - A URL that points to the WSDL file
   - A UDDI location

   This example uses a UDDI registry.

4. The request from the client is redirected to the service provider (for example, Wholesale1).

5. The response is received by the gateway service.

6. The response is sent through the channel to the client.

**Stage 2: Enhanced integration**

In stage 2, a filter is added to the Router to determine the appropriate service needed to fill the request. The Retail system no longer needs to know which Wholesale supplier provides the part.
The solution using WebSphere Web Services Gateway in this stage is shown in Figure 11.

1. The client application (Retail) connects to the gateway using SOAP/HTTP to send the request.
2. The service invoked by the client (for example, InventoryWholesale) receives the request and resends it to the request filter associated with it. The filter is actually an EJB in an enterprise application.
3. The component that implements the filter receives the request. In this example, the filter looks at the content of the message and selects the target service to be invoked.
4. The request sent by the client is sent to the target service specified by the filter.
5. The location of the service provider (for example, Wholesale1) is obtained from the WSDL associated with the service.
6. The request from the client is redirected to the service provider (for example, Wholesale1).
7. The response is received by the gateway service.
8. The response is sent through the channel to the client.
Stage 3: Extended enterprise

The last stage is to make the Wholesale services available to external Retail systems.

Stage 3 builds on stage 2 by externalizing the Wholesale services. The considerations are primarily security concerns. The interaction between the Retail, Wholesale, and Web Services Gateway remain the same as in stage 2.

Development guidelines

Normally, when implementing a Router solution, the interface of the service is determined by an existing application (in ABC Electronics, the Wholesale department application). In other cases, you implement a Web service interface that encapsulates one or more operations of an existing application.
If you plan to implement a Router Pattern, all the service providers have to use the same service interface (targetNamespace, operations, etc.) and the same binding (RPC or Document, literal or encoded, etc.). If there is any difference between the service interface of the different providers, you will have to implement a request filter that transforms the incoming message to the service provider format.

Before implementing the Router solution, confirm that both the client and Web service applications are working properly. For example, try connecting them directly, without going through the gateway.

**Implementation process**

The section outlines the procedures used to implement the solutions for the ABC Electronics scenario.

**Note:** This process assumes the following conditions:

- An enterprise application called WSWholesale1EAR exists. It contains an EJB module with an EJB called WSWholesale1EJB. The EJB is coded to return a delivery date for a part number supplied by the client.
- A second enterprise application called WSWholesale2EAR also exists. It is similar to the first Wholesale application.

This section will describe the process used to create Web services from the two EJ Bs, register them to the UDDI Registry and the Web Services Gateway, and create a client to access the newly created Web services.

This process also assumes that:

- A third enterprise application called WSRetailEAR has been created and skeleton code has been added. The only thing missing is the code that invokes the Web services through the Web Services Gateway.

Here is a summary of the necessary steps:

1. Web-service-enable the target application. The solution assumes that the Wholesale systems have implemented their delivery date applications using an EJB which can easily be converted to a Web service.

2. Register the business and services in the UDDI registry. This step is the same regardless of the stage you are implementing, however, in stage 1 and 2 (intra-enterprise) you might elect to use a private UDDI registry. In stage 3, you might consider using a commercial UDDI registry.
3. Create and deploy the filter application (stages 2 and 3).
4. Create the Web Services Gateway services and export the WSDL.
5. Export the Web services definition from the Web Services Gateway.
6. Web service enable the source application using the WSDL exported from the Web Services Gateway. The Retail system needs to have the proper information to call the Web services.
7. Test the source and target applications in the WebSphere Studio test environment.
8. Deploy the source and target applications in your WebSphere Application Server runtime environment.

**Web service enable the target application**

Figure 13 shows the steps involved in creating the Web services enabled target application using the stateless session bean from our sample code.

![Figure 13 Web service development for target application](image)

Let's walk through the process shown in Figure 13 for our target application.
We have already created a simple EJB application using IBM WebSphere Studio Application Developer that looks like the structure shown in Figure 14.

![Figure 14 Wholesale application](image)

**Note:** This discussion assumes two existing applications for the Wholesale function, called WSWholesale1EAR and WSWholesale2EAR. Each has an EJB module (WSWholesale1EJB and WSWholesale2EJB). For this exercise, they both run on the same host at http://wholesale1:9081.

These directions work for both applications. We use WSWholesale1EAR as an example. To do this for Wholesale2, just substitute Wholesale2 wherever you see Wholesale1.

Testing the EJB using WebSphere Studio:

A simple method of testing an EJB deployed to the WebSphere Studio test environment is to use the Universal Test Client. After you have deployed the new application to a server and started the server, simply do these steps:

1. Select the server, right-click, and select **Run universal test client**.
2. When the test client starts, click the **JNDI Explorer** link. Expand the **ejb** and folder so you can see a reference to the EJB home interface. Click this link.
3. To create an instance of the bean, expand EJB in the References window until you see the **create()** method for the EJB. Click this link.
4. In the Parameters window click **Invoke**, then **Work With Object**. An instance will be added in the References window. The new name will be **EJBname_1**.
5. Expand the new EJB instance to list the methods. To test a method, click it and a window will open allowing you enter input parameters and invoke the method. The results will be shown in the same window.
We can create an RPC style Web service using this EJB as follows:

1. Create a service endpoint interface for the Web service in a new package.
   a. Using WebSphere Studio, switch to the J2EE Perspective and click the **Project Navigator** tab.
   b. Navigate to the **WSWholesale1EJB → ejbModule** folder.
   c. Right-click the **ejbModule** folder and select **New → Package**. Set the package name to **com.ibm.itso.wholesale.inventory**. Click **Finish** to create the package.
   d. Create the service endpoint interface by copying the Inventory EJB remote Interface, Inventory.java, from the com.ibm.itso.ejb.inventory package to Inventory.java in the new com.ibm.itso.wholesale.inventory package.
   e. Edit the new Inventory.java file, so that it extends java.rmi.Remote, as shown in Example 1.

   **Example 1  Inventory service endpoint interface**

   ```java
   package com.ibm.itso.wholesale.inventory;
   public interface Inventory
       extends java.rmi.Remote
   {
       public String getDeliveryDate(String partNo)
           throws java.rmi.RemoteException;
   }
   ```

2. Open a command window and change to WSWholesale1EJB \ejbModule in your Studio workspace folder.

3. Generate a Web Services Description Language (WSDL) file from the service endpoint interface. We used the command shown in Example 2. Note that this command assumes we will use /Wholesale1 as the root context and services/InventoryWholesale1 to access the EJB.

   **Example 2  Generating WSDL using Java2WSDL**

   ```bash
   c:\WebSphere\appserver\bin\java2wsdl -verbose
   -implClass com.ibm.itso.ejb.inventory.InventoryBean
   -location http://wholesale1:9081/Wholesale1/services/InventoryWholesale1
   -output c:\Workspace\WSWholesale1EJB\ejbModule\META-INF\wsdl\Wholesale1Direct.wsdl
   -style rpc -use literal -voidReturn ONEWAY com.ibm.itso.wholesale.inventory.Inventory
   ```

4. Using the WSDL file created in the previous step, generate the Web services deployment descriptors and classes using the WSDL2Java tool. We used the command shown in Example 3.
Example 3  Generating server deployment descriptors and classes using WSDL2Java

C:\WebSphere\AppServer\bin\WSDL2Java -verbose -role server -container ejb -output C:\Workspace\WSWholesale1EJB\ejbModule
c:\Workspace\WSWholesale1EJB\ejbModule\META-INF\wsdl\Wholesale1Direct.wsdl

Note: The deployment descriptors and class files will not be regenerated when the tool is re-run, unless the existing files have been removed first.

5. Right-click the EJB project and select Refresh. You should see the generated files shown in Figure 15.

The webservices.xml deployment descriptor defines the set of Web services that are being deployed in the Web service-enabled J2EE container.

Figure 15  Generated files

6. Navigate to WSWholesale1EJB → ejbModule → META-INF and edit webservices.xml. Set the ejb-link element to Inventory, as shown in Example 4.

The ejb-link element corresponds to the ejb-name element of the required EJB, as defined in ejb-jar.xml.

Example 4  Updating webservices.xml

...  
  <service-impl-bean id="ServiceImplBean_1063202220893">  
    <ejb-link>Inventory</ejb-link>  
  </service-impl-bean>  
...
7. Run the endptEnabler command line tool to add an HTTP router module to the EAR file.

**Tip:** This step is only required if the Web service is implemented in an EJB module.

In WebSphere Studio V5.1, right-click the EAR project, and select **Web Services → Endpoint Enabler** from the pop-up menu.

**Note:** Alternatively, you could also use the command shown in Example 5. To do this you first need to export the WSSholesale1EAR project to an EAR file:

a. Right click the **WSSholesale1EAR** project and select **Export**....
b. Select **EAR file** and click **Next**.
c. Select the destination you want the EAR file to be exported to, for example:
   
   C:\WebSphere\AppServer\installableApps\WSSholesale1EAR.ear
d. Click **Finish**.
e. Run the endptEnabler command:

   C:\WebSphere\AppServer\bin\endptEnabler

   WSSWS2004I: IBM WebSphere Application Server Release 5
   WSSWS2006I: Please enter the name of your EAR file: WSSholesale1EAR.ear
   WSSWS2003I: Backing up EAR file to: WSSholesale1EAR.ear~
   WSSWS2016I: Loading EAR file: WSSholesale1EAR.ear
   WSSWS2017I: Found EJB Module: WSSholesale1EJB.jar
   WSSWS2029I: Enter http router name for EJB Module WSSholesale1EJB
   [WSSholesale1EJB_HTTPRouter.war]: WSSholesale1EJB_HTTPRouter.war
   WSSWS2030I: Enter http context root for EJB Module WSSholesale1EJB
   [/WSSholesale1EJB]: Wholesale
   WSSWS2024I: Adding http router for EJB Module WSSholesale1EJB.jar.
   WSSWS2036I: Saving EAR file WSSholesale1EAR.ear...
   WSSWS2037I: Finished saving the EAR file.
   WSSWS2018I: Finished processing EAR file WSSholesale1EAR.ear.
The endptEnabler tool makes the following changes to the EAR file:

- Adds a Web module to the EAR file that contains the HTTP router for the EJB module and sets the context root for the Web module in application.xml.

**Note:** If you used the WebSphere Studio Endpoint Enable wizard, the context root for the Web module is automatically chosen for you. Open the EAR deployment descriptor and if necessary, alter the context root to match the context you used earlier in the -location attribute of the Java2WSDL command (see Example 2 on page 24. In this example, the context is /Wholesale1.

- Adds servlet and servlet-mapping elements to the Web module deployment descriptor. These elements map the Web service endpoint URL to the Web services router servlet, and are added for each Web service in the module.

**Note:** The URL mappings must match the values you used in the -location attribute of the Java2WSDL command (see Example 2 on page 24) — in this example, “services/InventoryWholesale1”.

![Application Deployment Descriptor](image)
Add a routerModule element to ibm-webservices-bnd.xmi in the EJB module.

The EAR file is now ready to deploy in the IBM WebSphere Application Server V5.0.2 runtime.

8. Generate the EJB deployment code:
   a. Select, then right-click the WSWholesale1EAR project, and select Generate → Deploy and RMIC Code from the pop-up menu.
   b. In the Generate Deploy and RMIC Code window, click Select all to select all the EJBs, then click Finish.

Testing the Web service using WebSphere Studio:

A simple method of testing the Web service you have created is to use the Web Services Explorer. To use the Web Services Explorer, right-click the WSDL file for the Web service and select Web Services -> Test with Web Services Explorer. The Explorer will start in a Web browser window. From there you can select the Web service method you want to test, or add / change the Web service endpoints.

Register the services in the UDDI Registry

In this scenario, the private UDDI Registry shipped with the WebSphere Application Server Network Deployment and Enterprise packages was used. To make Web services available through the private UDDI registry, follow these steps:
1. Install the UDDI Registry. The UDDI Registry used in this scenario was installed from and deployed on WebSphere Application Server Enterprise V5.0.2 with the 5021 fix applied.

Once the application server and the Web services features have been installed and brought up to the correct fixpack level, you will need to complete the installation of the UDDI Registry using the instructions in the InfoCenter.

The UDDI Registry is an enterprise application that runs on an application server, so this process includes creating a database for the UDDI Registry to store data in and installing the UDDI Registry application. Scripts are provided to do this.

2. Register the business.

3. Add a technical model of the Web service.

4. Add the service to the business, associating the correct technical model to the service.

**Register the business**

The first step in setting up the UDDI Registry is to publish the business in the registry.

1. Open the UDDI registry console:

   (http://uddi.itso.ral.ibm.com:9080/uddigui/)

2. Select the Publish tab and click **Add a business** under the Advanced Publish menu.

3. Enter the business name, select a language, and click the **Add** button to the right of the Language field. This allows you to add multiple names for the same business. It doesn’t actually publish the business yet. That doesn’t happen until you click the **Publish Business** button, once you have completed the rest of the fields.

4. Add a description if you like and click the **Add** button to the right.
Figure 16  Register the business

5. Click **Publish Business**. This causes the business to be published to the UDDI Registry, and a page is displayed showing the business details.

**Add a technical model**
Add a technical model (tModel) for Wholesale 1 by doing these steps:

1. Select the Publish tab and click **Add a technical model** under the Advanced Publish menu.

2. Enter the following values: (where there is a blue Add link, you must click that to add the data you enter in the fields to the left).
– Name: InventoryWholesale1
– Description: Inventory of Wholesale 1 (click Add)
– Locator (describe the technical model with taxonomies)
  • Locator Type: unspsc
  • Locator Key name: (blank)
  • Locator Key value: wsdlSpec
  (Click Add)
– Overview URL (a URL pointing to an overview document, a description of the document, and a Language field):
  http://wholesale1:9081/Wholesale1/services/InventoryWholesale1/wsdl/Wholesale1Direct.wsdl

![IBM WebSphere UDDI Registry user console - Microsoft Internet Explorer](image)

**Figure 17 Add a technical model**

3. Click the **Publish Technical Model** link to create the technical model in the UDDI Registry.
Add a service to the business

Follow these steps:

1. Select **Show owned entities** under the Publish tab.
2. Locate the business and click the **Add service** button.
3. Enter the relevant data (click **Add** where applicable):
   - Service Name: InventoryWholesale1
   - Description: Inventory Service of Wholesale1
   - Access points (points to the Service):
     - URL Type address:
       wholesale1:9083/Wholesale1/services/InventoryWholesale1/wsd1/Wholesale1
       Direct.wsdl
   - Locator (add references to taxonomies to the service)
   - Technical Model (associate existing tModels to the Service):
     InventoryWholesale1

4. After completion of those areas required, click the **Publish Service** button to publish the service to the UDDI Registry with the current form contents.

Figure 18
Create the filter

A filter is simply an EJB running in an enterprise application. To create the filter, do the following steps with WebSphere Studio:

1. Create an enterprise application. (ITSORouterApp in the scenario).
2. Create an EJB project. (ITSORouterAppEJB in the scenario).
3. Create a session bean. (RouterBean in the scenario). The easiest way is to use the J2EE Hierarchy view in the J2EE perspective. Select the **Session Beans** folder, right-click, and select **New->Session Bean**. The values we used are shown in Figure 19 and Figure 20.

![Create a session bean for the filter](image)

*Figure 19  Create a session bean for the filter*
When you create an EJB filter, you have to use the following interfaces:

- com.ibm.wsgw.beans.FilterHome
- com.ibm.wsgw.beans.FilterRemote

**Tip:** To develop the filter, you need to have the wsgwejb.jar, which comes in the `<WAS_Enterprise_HOME>/lib` directory in your build path.

**Important:** The support of filter programming is supported only in IBM WebSphere Application Server - Enterprise.
Figure 21 shows the results of the Enterprise Bean wizard.

Figure 21   Developing a filter with WebSphere Studio

A filter can get the list of potential target services from the Web Services Gateway Routing service. The Home object for this service must implement the com.ibm.wsgw.beans.RoutingHome interface and be located in JNDI at websphere/WSGW/Routing.

The sequence of events is as follows:

1. The filter is called with a WSIFRequest.
2. The filter obtains the list of potential target services from the Routing service.
3. The filter selects the target service.
4. The filter calls the Routing service to set the target service (note that doing this clears any prior selection of a target service's port).

Each target service is identified by its unique target service definition location and target service identity information (which might not be unique). To select the target service, your filter can either get the table of mappings from the target service location to identity information, then choose a target service to use; or it can call setSelectedTargetServiceIdentity with the required identity string (relying on the target service identity information being carefully defined).
The routing service then selects the first target service it finds (for the current gateway service) with identity information matching that specified (using String.equals()).

**Note:** When you use Routing to set the target service or the target port, the Routing service updates the request context. Because the request context has changed, you then need to set the request object in the FilterAction object that you return from the filterRequest method.

The filter can also use the Routing service to select the target port for the service invocation.

Example 5 shows how to determine the target service using a filter.

`Example 5  RouterBean.java`

```java
public FilterAction filterRequest(WSIFRequest req, WSIFResponse resp)
    throws FilterException, WSGWException {
    try {
        InitialContext initialContext = new InitialContext();
        RoutingHome routingHome =
            (RoutingHome)PortableRemoteObject.narrow(
                initialContext.lookup("websphere/WSGW/Routing"), RoutingHome.class);
        RoutingRemote routing = routingHome.create();
        WSIFMessage message = req.getIncomingMessage();
        String partNumber = ((JROMStringValue)message.getObjectPart("partNo")).getValue();
        if (Integer.parseInt(partNumber)<100) {
            routing.setSelectedTargetServiceIdentity(req, "InventoryServiceWholesale1");
        } else {
            routing.setSelectedTargetServiceIdentity(req, "InventoryServiceWholesale2");
        }
    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
    return null;
}
```

Note that the target services identified here haven’t been defined to the Web Services Gateway yet. Later, when you define them, the target service identity information (see Figure 26 on page 42) must match with the filter code.
Deploy the filter
To deploy the filter, you have to install the enterprise application that contains the filter. Later, you will also need to configure the filter to the Web Services Gateway. Follow these steps:

1. Export the EAR file from WebSphere Studio to the <was_install>/installableapps directory.

2. Using the WebSphere administrative console for server1 (where the Web Services Gateway application is installed), select Applications->Install New Application.

3. Browse to the ear file (ITSORouterApp for our scenario), select it and click Next.

4. As you proceed through the application installation, the steps may vary depending on the features of the application (whether it uses security, a database, etc.). As you go through the install, make sure the JNDI name for the EJB is correct.

Refer to Figure 22.

![WebSphere Administrative Console](image)

**Figure 22** Installing the filter application - panel 3

4. Complete the installation and start the application.
Define the services and filter to Web Services Gateway

Before you define a Web service to the Web Services Gateway, deploy the resources (channels, filters, UDDI references and security bindings) that the Web service uses.

1. Install the Web Services Gateway. The Web Services Gateway used in this scenario was installed from and deployed on WebSphere Application Server Enterprise V5.0.2 with the 5021 fix applied.

Once the application server and the Web services features have been installed and brought up to the correct fixpack level, you will need to complete the installation of the Web Services Gateway using the instructions in the InfoCenter.

The Web Services Gateway is an enterprise application that runs on an application server, so this process includes creating a database for the Web Services Gateway to store data in, and installing the Web Services Gateway applications. Scripts are provided to do this.

2. Perform the initial configuration of the gateway, including the namespace URI. If you change the namespace URI, you break the link back to the Web Services Gateway for every Web service that you have already deployed. So you must set the namespace URI before you deploy any Web services.

3. Add a channel to handle incoming SOAP requests.

4. Add a reference to the UDDI Registry.

5. Deploy the Web services to the gateway. If you aren’t using a filter to route service requests, you will need to create one service for each service provider (InventoryWholesale1 and InventoryWholesale2). If you are using a filter to route requests, you will only need to create one service, with two target services (InventoryWholesale1 and InventoryWholesale2).

Initial configuration

1. Open the Web Services Gateway administration console at:

   http://wsgw_host:9080/wsgw/admin/

2. Select the Configure option under Gateway.

3. Enter the following values (Figure 23):
   
   - Namespace URI for services: urn:ibm:wsgw. (default)
   - WSDL URI for exported definitions: http://wsgw_host:port/wsgw
4. Click **Apply Changes**.

**Add a channel**

To deploy a channel for incoming SOAP requests (Figure 24):

1. Select **Deploy** under the Channels menu option.

2. Enter the following values:
   
   - Channel Name: SOAPHTTPChannel1
   - Home Location: SOAPHTTPChannel1Bean
   - End Point Address: http://wsgw:9080/wsgwsoaphttp1
3. Click **OK**.

**Add a UDDI reference**

To deploy a UDDI Reference (Figure 25):

1. Select **Deploy** under the UDDI References menu option.

2. Enter the following values:
   
   - **Name**: UDDIPrivateRegistry
   - **inquiryURL**: http://uddi:9080/uddisoap/inquiryapi
   - **publishURL**: http://uddi:9080/uddisoap/publishapi
   - **userName**: authorized user ID
   - **password**: 
3. Click **OK**.

**Deploy the Web services**

To deploy a Web service (Figure 26), follow these steps:

1. Select **Deploy** under the Services menu option.

2. Enter the appropriate values for the service. In this scenario, the values used for the InventoryWholesale1 service were:
   - **Gateway Service Name**: InventoryWholesale1
   - **Message part representation**: Generic classes
   - **Channels**: SOAPHTTPChannel1
   - **UDDI References**: UDDIPrivateRegistry
Figure 26  Deploying a service - part 1

- WSDL Location: `uddiReference, serviceKey`
  
The service key can be obtained in the UDDI Registry by switching to the Publish tab and selecting:

  **Show owned entities -> ABC Electronics/Show services -> InventoryWholesale1.**

- Location Type: UDDI

- Target Service Name: InventoryService

- Target Service Namespace: `http://inventory.wholesale.itso.ibm.com`

- Target Service Identity Information: `InventoryServiceWholesale1`

- Business Key: Business Key (for example, BC01B8BC-7A9F-4F56-86FE-61AB5D61FE75)
  
The business key can be obtained in the UDDI Registry by switching to the Publish tab and selecting:

  **Show owned entities -> ABC Electronics/Show services -> InventoryWholesale1**
Note that the target service identity information matches the code in the filter (see Example 5 on page 36).

**Note:** If you are not using a UDDI Registry, you will need to make the WSDL file available to the gateway. A simple way to do this is to create a new folder for the WSDL files in the application folder for the Web Services Gateway application. For example:

```
<WAS_HOME>/installedApps/<node_name>/wsgw.server1.<node_name>.ear/wsgw.war/wsdl
```

Then copy the WSDL file from the WebSphere Studio workspace to the new folder.

When defining the service, simply select URL as the location type and enter the URL for the wsdl file. In this case, the URL would look like:

```
http://<wsgw_node>:9080/wsgw/wsdl/<wsdl_file>.wsdl
```

Repeat the process to deploy each service. For our stage1 application, both InventoryWholesale1 (Figure 28) and InventoryWholesale2 (Figure 29) need to be deployed.
In stage 2, only one service is deployed (InventoryWholesale) but has two target services. To create a service with multiple target services, create a new service using the method just described.

During the process you will add the first target service. To add two service targets to the same service definition, you first create a service and specify one of the services as the target (for example, Wholesale1). Once the service is created, you can list the service and add a second target service (Wholesale2).

*Figure 28 Service for Wholesale1*

Then, after creating the service, add the second Web service target.
3. Click **Apply Changes** to update the service (Figure 30).
Configure the filter
To configure the filter (Figure 31), follow these steps:

1. Select the **Deploy** option under the Filters menu.

2. Enter the filter name and the home location (JNDI name) of the EJB Filter and click **OK**. The home location is the same as specified during the filter install (see Figure 22 on page 37).
Figure 31  Deploy the filter to the gateway

3. Find the service you want to add the filter to (Wholesale) by selecting **List** under the Services option, then selecting the service.

4. Add the filter to the service by selecting the filter name in the Request Filter field.

   You can add multiple filters to a service (Figure 32). The position selected determines the order in which the filters are executed. In this instance, there is one filter, so the position is “At the end”. Click **Add**.

Figure 32  Adding a filter to a service

**Export the Web service definition**

The Web Services Gateway dynamically generates WSDL files for the Web services it is serving (one file for the interface and one file for the implementation). You have to copy this definition to build your client application (in our case, the Retail application).
In the Exported WSDL Definitions section, there are two pairs of WSDL links. Both pairs link to (a) the external WSDL implementation definition, and (b) the external WSDL interface definition:

- To view details of the associated external WSDL for the service, use the first pair.
- To return the WSDL for use by application programs that need the WSDL definitions for the service, use the second pair.

If there is an error generating the WSDL, then a blank page is returned.

**Note:** To help your service users locate the WSDL documents for services that are deployed to the Web Services Gateway, the gateway also supports the WS-Inspection specification. To open a WS-Inspection document which contains references to the WSDL documents for all of the gateway-deployed services, you issue an HTTP GET against:

```
http://host:port/wsgw/wsinspection.wsil
```

Here, *host* and *port* are the host name and port number that your HTTP server is listening on.

**Export the WSDL from the Web Services Gateway**

From the Web Services Gateway admin console, follow these steps:

1. You will need a place in the application to store the WSDL files (Figure 33). From WebSphere Studio, create a new folder called wsdl under WSRetailWeb/WebContent/WEB-INF.

2. Select the service (for example, InventoryWholesale1) and right-click **External WSDL implementation definition**.
Figure 33  View the WSDL

Select **Save Target As..** and save the file as **InventoryWholesale1WSGW-Impl.wsdl** file in the **WSRetailWeb/WebContent/WEB-INF/wsdl** folder (Figure 34).
3. Follow the same steps to save the external WSDL interface definition as InventoryWholesale1WSGW-Interface.wsdl.

4. Repeat the process to create the InventoryWholesale2WSGW-Impl.wsdl and the InventoryWholesale2WSGW-Interface.wsdl files.

**Web service-enable the source application**

Figure 35 shows the steps involved in Web services-enabling the source (client) application.
Let’s walk through the process shown in Figure 35 for our source application.

When developing a Web services client, you must have access to a server application. This could be developed by a separate department within your organization, which will supply the WSDL file defining their server. In our example, we use the target application described in, “Web service enable the target application” on page 22.

Web service-enabling our source application is simply a matter of generating the required deployment descriptors and proxy classes from the gateway-generated WSDL files for the target application.

**Note:** If you are connecting directly to the Web service without going through the Web Services Gateway, the WSDL comes directly from the target system. If you are using a Web Services Gateway, the WSDL is created and stored at the Gateway.

1. In the previous process, you exported the WSDL files from the Web Services Gateway to your source application. Before proceeding it is a good idea to test the operation of the Web service, both directly and through the gateway. It is simple to do this using the Web Services Explorer in WebSphere Studio.
   a. Right-click the -Impl WSDL file in the source application, and select **Web Services → Test with Web Services Explorer** (Figure 36).
   b. Select **getDeliveryDate** and enter a part number.
   c. Click **Go**.
Figure 36  Using the Web Service Explorer to test the Web service

Make sure the results are correct. If there are any problems, repeat the process using the WSDL from the target application to see if the problem lies with the gateway definitions or with the Web service itself.

Once you are sure the Web service is working, continue by creating the client code.

2. Using WebSphere Studio, switch to the J2EE Perspective and click the **Project Navigator** tab. Navigate to the **WSRetailWeb -> WebContent -> WEB-INF -> wsdl** folder.

3. Select the WSDL file, right-click and select **Web Services -> Generate Client.**

**Note:** The WSDL files determine how the Web service will be accessed. If you are accessing the service directly, you will have a single WSDL file generated by the target application. If you are using a Web Services Gateway, you will select the implementation WSDL file generated at the gateway. Our sample application allows the user to select whether to access Wholesale1 or Wholesale2 directly, using the gateway, or via the filter. The following process illustrates the client generation using the WSDL for Wholesale1 generated at the gateway.
d. Select **Java Proxy** as the proxy type (default) and click **Next**.

e. Select **Use Defaults** for the client-side environment selection and click **Next**.

f. Make sure the correct WSDL file is selected. If you are using WSDL files generated at the Web Services Gateway, select the implementation WSDL file. It has a reference to the interface WSDL file.

g. Take the defaults on the next panel (generate proxy) and click **Finish**. The generated files are highlighted in Figure 37.

4. In our sample application, we will eventually offer several options to the user. These options allow the user to select Wholesale1 or Wholesale2 either directly or via the gateway. In order to do this, we repeated this process for each option and as each client code portion was generated, we renamed the classes to a unique name. To do this, we used the rename option for the class and selected each option beneath. This ensured that every reference to the class was changed, including those in the binding files (Figure 38).
5. Note that the webservicesclient.xml deployment descriptor defines the JNDI name for accessing the Web service and the associated service endpoint interface to be used. This descriptor is generated automatically by the Web Services Client wizard (Figure 39).
6. Add client application code to invoke the Web service on the target application.

   To invoke getDeliveryDate on the target application, we added the code shown in Example 6. We added this code to the com.ibm.itso.command.WSGWW1WebServiceBean command bean in our WS RetailWeb module.

   Example 6  Web service client code for getDeliveryDate

```java
package com.ibm.itso.command;
import ibmwsgw.Wholesale1;
import java.net.URL;
import javax.naming.*;
import com.ibm.itso.wholesale1.inventory.Inventory;

public class WSGWW1WebServiceBean extends CommandBean {

    ...

    public String getDeliveryDate(String partNumber) throws Exception {
        String deliveryDate = null;
```
try {

    InitialContext initialContext = new InitialContext();
    Wholesale1 inventoryWholesale1 = (Wholesale1) initialContext.lookup("java:comp/env/service/Wholesale1");

    //Request the Service Endpoint from the Service
    Inventory inventory = inventoryWholesale1.getInventorySOAPHTTPBindingPort();

    //Get the quote
    deliveryDate = inventory.getDeliveryDate(partNumber);

} catch (Exception e) {
    writeToLog("Exception: " + e.toString());
    throw new Exception(e.toString());
}

writeToLog("Exit - getDeliveryDate");
return deliveryDate;
}

---

Changing the Web service endpoint URL

By default, the endpoint proxy class (InventoryServiceLocator in our example) will use the endpoint address specified in the WSDL file when the proxy class was originally generated. It is likely that the endpoint address used will change over time, as the client application moves through the development, test, and production environments, for example. You can set the endpoint address in several ways:

1. When getting an instance of the generated endpoint class from the Service interface, you can optionally pass an endpoint address that will override the default address obtained from the WSDL file. Compare the following approach with that used in Example 6 on page 55:

   // Request the Service Endpoint from the Service
   // overriding the default endpoint address
   Inventory inventory = inventoryService.getInventory(new java.net.URL("http://localhost:9080/WSWholesale1EAR/services/Inventory"));

2. When deploying your client application to WebSphere Application Server, you can specify the **Deploy WebServices** option, as shown in Figure 40.

   WebSphere will regenerate the deployment code based on the Web services client deployment descriptors, updating it with the current endpoint address from the WSDL file. Similar to EJB deployment, this only needs to be performed when the deployment details have changed.
Example 7 shows how the endpoint address is specified in the WSDL file for our Inventory service.

The second option is recommended in most cases. Using this method, the endpoint address can be specified at application packaging/deployment time using the Web services deployment features of the application server.

**Example 7  Setting the endpoint address in the WSDL file**

```xml
...<wsdl:service name="InventoryService">
    <wsdl:port name="Inventory" binding="intf:InventorySoapBinding">
        <wsdlsoap:address location="http://localhost:9080/ITSOWholesale1/services/Inventory"/>
    </wsdl:port>
</wsdl:service>
...
```

**Best practices**

This section covers best practices and recommendations.

**Place all of the WSDL in one file**

WebSphere Studio V5.0 creates three files for the WSDL (interface, binding, implementation). WebSphere Studio V5.1 puts the WSDL into a single file. If you are using WebSphere Studio prior to V5.1, it is a good idea to combine the
WSDL into one file to eliminate any problems the gateway might encounter parsing include statements.

**Handling SOAP fault messages**

Example 8 shows the recommended way to create and return a SOAP fault message from the GatewayFilter.filterResponse method.

```java
public FilterAction filterResponse(WSIFRequest wsifRequest, WSIFResponse wsifResponse)
    throws FilterException, WSGWException, RemoteException {
    // Construct the fault message
    WSIFMessage faultMessage = new WSIFDefaultMessage();
    faultMessage.setObjectPart(WSIFConstants.SOAP_FAULT_ACTOR, "mySoapFaultActor");
    faultMessage.setObjectPart(WSIFConstants.SOAP_FAULT_CODE, "mySoapFaultCode");
    faultMessage.setObjectPart(WSIFConstants.SOAP_FAULT_STRING, "mySoapFaultString");
    faultMessage.setObjectPart("stackTrace", "myStackTraceDetails");
    faultMessage.setObjectPart("otherDetails", "myOtherDetails");
    // repeat faultMessage.setObjectPart("aaaa", "bbbb"); for each additional detail element

    // Set the fault message into the wsifResponse object
    wsifResponse.setFaultMessage(faultMessage);
    wsifResponse.setIsFault(true);

    // Return the updated response in the filterAction object
    FilterAction filterAction = new FilterAction();
    filterAction.setUpdatedResponse(wsifResponse);
    filterAction.setContinueProcessing(false);
    return filterAction;
}
```

**Handling exceptions for the Web Services Gateway**

During normal processing of a Web service invocation, a fault message might be generated by the target service, and is passed back to the channel to be sent to the originator. As far as the Web Services Gateway is concerned there is no difference between processing a normal output message and processing a fault message.

But when an exception occurs during processing of a request, the channel needs some way to decide what to do with the exception. What is needed is a service that provides a pluggable handler that can look at the message, exception and other information to decide whether the exception should be thrown back to the originator, or whether a fault message should be constructed.
This service is not provided with the Web Services Gateway, but the gateway does contain an interface to encapsulate such a service. The ExceptionHandler interface allows channels to call an exception handling service, and allows the exceptions to be reported to a third party for analysis.

The Home object for this service must implement the com.ibm.wsgw.beans.ExceptionHandlerHome interface and be located in JNDI at websphere/WSGW/ExceptionHandlerService.

**Capturing Web service invocation information**

The Web Services Gateway has not implemented a service that stores operational messages, but it does contain an interface (the MessageWarehouse interface) to encapsulate such a service. This interface is driven by channels on receipt of requests and before sending responses.

If you have your own system for handling (classifying, storing and retrieving) operational messages, you can potentially use it to log the gateway's operational messages through the MessageWarehouse interface.

The Home object for this service must implement the com.ibm.wsgw.beans.MessageWarehouseHome interface and be located in JNDI at websphere/WSGW/MessageWarehouse.

**Monitoring SOAP messages**

You can trace the XML messages exchanged between a Web service client and the server. In this section we look at two tools:

- The TCPMon tool provided with IBM WebSphere Application Server V5.0
- The TCP/IP Monitor Server provided with WebSphere Studio Application Developer

**WebSphere TCPMon tool**

The TCPMon tool allows SOAP messages to be traced by redirecting messages from one port to another, displaying the contents as they go. The WebSphere application server normally listens on port 9080. To trace messages sent to the application server, TCPMon can be configured, for example, to listen on port 9088 and redirect messages to 9080. The client is modified to use port 9088 to access the Web service.

This tool is provided with IBM WebSphere Application Server V5.0.2. It allows you to view the contents of the SOAP messages exchange between the source and target applications, as shown in Figure 41.
You can start TCPMon from a command window as follows:

```
set CLASSPATH=%CLASSPATH%;<WAS_HOME>\lib\webservices.jar
<WAS_HOME>\java\bin\java com.ibm.ws.webservices.engine.utils.tcpmon
```

For further details on TCPMon, see the InfoCenter article *Tracing Web services messages* at:

The SOAP request for our getDeliveryDate() method is shown in Example 9.

Example 9  SOAP request for getDeliveryDate

POST /WSWholesale1EAR/services/Inventory HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: IBM WebServices/1.0
Host: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 427

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<getDeliveryDate xmlns="http://inventory.wholesale.itso.ibm.com">
<partNo xmlns="">12345</partNo>
</getDeliveryDate>
</soapenv:Body>
</soapenv:Envelope>

The SOAP response for the getDeliveryDate() method is shown in Example 10.

Example 10  SOAP response for getDeliveryDate

HTTP/1.1 200 OK
Server: WebSphere Application Server/5.0
Content-Type: text/xml; charset=utf-8
Content-Language: en-US
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<getDeliveryDateResponse xmlns="http://inventory.wholesale.itso.ibm.com">
<getDeliveryDateReturn xmlns="">09/12/2003</getDeliveryDateReturn>
</getDeliveryDateResponse>
</soapenv:Body>
</soapenv:Envelope>
TCP/IP Monitor Server
The TCP/IP Monitor Server provided with WebSphere Studio Application Developer (shown in Figure 42) also allows tracing of SOAP messages. It works in a similar way to the WebSphere TCPMon tool. To use the TCP/IP Monitor Server, create a new Server and Configuration and select Other → TCP/IP Monitor Server for the server type.

![TCP/IP Monitor Server](image)

Figure 42 Tracing SOAP messages using WebSphere Studio TCP/IP Monitor Server

- Back up your configuration.

  The Web Services Gateway offers a very easy way to make a backup of the entire configuration of the gateway. This backup does not include the filter implementation, which resides in a separated Enterprise Application.

Quality of service capabilities
In this section we discuss Quality of Service capabilities and considerations specific to the Web Services Gateway.

Performance and availability
In this section we discuss considerations regarding performance and availability.

SOAP Caching
SOAP caching can significantly improve the performance of Web services. The SOAPAction HTTP header in the request is defined in the SOAP specification and is used by HTTP proxy servers to dispatch requests to particular HTTP servers. WebSphere Application Server dynamic cache can use this header in its cache policies to build IDs without having to parse the SOAP message.
**Application server clustering**

The Web Services Gateway is a J2EE enterprise application that runs on WebSphere Application Server. This means that in a Network Deployment environment, clustering of application servers can be used to improve performance and availability.

For information on WebSphere Application server clustering, see *IBM WebSphere V5.0 Performance, Scalability, and High Availability*, SG24-6198.

**Security**

Security is one of the crucial QoS aspects which needs to be addressed when an enterprise plans to expose their internal applications to partner organizations. The following topics address Web Services Gateway security features.

**Invocation of services using SSL**

As a request passes from one component to another, the opportunities for the interception and exposure of information increase and ultimately the overall security of a system directly relates to the weakest, or least secure, point. SSL can be used to secure connections between two endpoints.

WebSphere Application Server, and the Web Services Gateway support SSL connections.

**Basic authentication and authorization facilities**

Web Services Gateway provides a basic authentication and authorization mechanism based upon the broader security features of WebSphere Application Server.

Basic authentication can be applied at two levels:

- Gateway-level authentication
- Web service operation-level authorization

For gateway-level authentication, you set up a role and realm for the gateway on WebSphere Application Server's Web server and servlet container, and define the user ID and password that is used by the gateway to access the role and realm. You also modify the gateway's channel applications so that they only give access to the gateway to service requesters that supply the correct user ID and password for that role and realm.

**Note:** This means that gateway-level authentication must be enabled before you install any channels.
For operation-level authorization, you apply security to individual methods in a Web service. To do this, you create an enterprise bean with methods matching the Web service operations. These EJB methods perform no operation and are just entities for applying security. Existing WebSphere Application Server authentication mechanisms can be applied to the enterprise bean. Before any Web service operation is invoked, a call is made to the EJB method. If authorization is granted, the Web service is invoked. Your target Web service is protected by wrapping it in an EAR file, and applying role-based authorization to the EAR file.

Notes:
- If you want to enable operation-level authorization, you must first enable gateway-level authentication.
- If you want to change the default gateway-level authentication settings, you must do so before you install any channels.
- After gateway-level authentication has been enabled, filters have access to the requester's authentication information.

WS-Security

Because the Web Services Gateway deals with SOAP messages at the application level, you are introducing the potential for a security risk. Securing the message at the transport protocol level is not sufficient. WS-Security describes how to secure messages using XML encryption and XML digital signature and how to integrate these specifications into a SOAP message.

Web Services Gateway can be configured for the secure transmission of SOAP messages using tokens, keys, signatures and encryption in accordance with the Web Services Security (WS-Security) draft recommendation.

In a normal (non-gateway) WS-Security scenario, the flow is as shown in the following figure:
The client generates a request which is then handled by the client Web services engine. It reads the client security configuration and applies the security defined in the `ibm-webservicesclient-ext.xmi` file to the SOAP message. It gets additional binding information from the `ibm-webservicesclient-bnd.xmi` file (for instance, the location of a keystore on the file system).

On receipt of a SOAP message, the Web services engine on the server refers to the `*.xmi` files for the called Web service. In this case, the `ibm-webservices-ext.xmi` file tells the engine what security the incoming message must have (for example, that the body of the message must be signed). If the message does not comply, then it is rejected. The Web services engine verifies any security information, then passes the message on to the Web service being called.

On the response leg from server to client, the process is reversed. The Web service `*.xmi` files tell the Web services engine what security to apply to the response message, and the client `*.xmi` files tell the client engine what security to require in the response message.

When the gateway is introduced, the scenario is more complex. Essentially it can be thought of as two separate request/response invocations. Client to gateway and gateway to target service, as shown in the following figure:
In this scenario, the client and the Web service are unchanged, and still have the same security settings in their *.xmi files. However, the gateway is unsecured. Secure SOAP messages cannot travel through the gateway unchanged, and must be processed on receipt. So the gateway needs to act as the target service from the point of view of the client, and as the client from the point of view of the target service.

This scenario means that the security settings for the Web service need to be configured for the view of the service that the gateway presents to the client, and the security settings for the associated gateway target services (remember that there may be multiple target services deployed for a single gateway service) need to be configured with the security settings for the client.

WS-Security settings for the gateway are configured manually using the gateway administrative user interface.

**Proxy authentication for the gateway**

The gateway requires access to the Internet for invoking Web services and for retrieval of WSDL files. Many enterprise installations use a proxy server in support of Internet routing, and many proxy servers require authentication before they grant access to the Internet. This requirement is supported in HTTP messaging by a “Proxy-Authorization” message header that contains encoded username and password credentials.

For messages passing through the gateway, you can enable and disable proxy authentication, and specify whether the authentication credentials are supplied by the service requester or by the gateway. If you specify requester-supplied credentials, the credentials in the HTTP message that the gateway receives are re-instantiated by the gateway in the equivalent message that it sends on to the proxy. If you specify gateway-supplied credentials, the gateway ignores any credentials in the incoming HTTP message and supplies its own credentials in the equivalent message that it sends on to the proxy.
Standards compliance

By utilizing open standards, Web services can, in theory, enable any two software components to communicate regardless of what technologies or platforms are used to create or deploy them. Interoperability across heterogeneous platforms is one of the key value propositions of Web services.

**WS-I Profile**

The Web Services Interoperability Organization is an open industry effort chartered to promote Web services interoperability across platforms, applications, and programming languages. Web services for J2EE intends to conform to the WS-I Basic Profile 1.0, and should interoperate with any other vendor conforming to this specification.

**WS-Security**

You can configure the gateway for secure transmission of SOAP messages using tokens, keys, signatures and encryption in accordance with the Web Services Security (WS-Security) draft recommendation.

**Interoperability through firewalls**

The Web Services Gateway is entirely based on Web services, which when implemented using HTTP as SOAP transport protocol, are totally firewall friendly.

**Autonomic**

Log and trace facilities are important for fault monitoring and isolation. WebSphere Application Server provides a number of log files. JVM logs are located in the `<WAS_HOME>/logs/<applicationServerName>` directory, and by default are named SystemOut.log and SystemErr.log.

The Diagnostic Trace Service can be used to enable tracing of application server components. The following trace specification can be used when diagnosing Web Services Gateway problems:

```
com.ibm.wsgw.*=all=enabled:
org.apache.wsif.*=all=enabled:
com.ibm.ws.webservices.*=all=enabled
```
The following tools can also be helpful when analyzing problems:

- **TCPMon:**
  
  The TCPMon tool allows you to view the contents of the SOAP messages being generated by the interaction between the source application, gateway, and target application.

- **Tivoli® Performance Viewer:**
  
  The Tivoli Performance Viewer packaged with IBM WebSphere Application Server V5.0 can monitor Web Services Gateway requests and responses. When the gateway is installed in WebSphere V5.0, counters are added automatically to the Performance Monitoring Service. Gateway monitoring can be enabled by simply starting performance monitoring. See the redbook for details on specific procedures for enabling monitoring:

  – *IBM WebSphere Application Server V5.0 System Management and Configuration: WebSphere Handbook Series, SG24-6195*

Once monitoring is enabled on the WebSphere Application Server, open the Tivoli Performance Monitor and navigate to the Web Services Gateway. The Viewer monitors both synchronous and asynchronous requests and responses. The output can be viewed in either table or graph format. It can also be logged and played back when needed. Figure 45 shows the Tivoli Performance Viewer monitoring gateway requests and responses.

![Figure 45  Tivoli Performance Viewer](image)
Microsoft Network Monitor:

Microsoft Network Monitor captures network traffic on local area networks for real-time or post-capture analysis. The Network Monitor captures frames from the network that can be filtered to present only relevant material. It can also be configured to detect specific network conditions and generate events as needed. Figure 46 shows the Microsoft Network Monitor. Details concerning the configuration and use of the monitor can be found from Windows Help.

Of particular interest to the gateway is the Network Monitor’s ability to monitor TCP/IP packets from a particular target and source. In this manner, traffic to and from the gateway can be traced from its origin to its destination.

![Microsoft Network Monitor](image)

*Figure 46  Microsoft Network Monitor*
References

- *Accelerate your Web Services with Caching*, Brian D. Goodman, December 2002
- *Best Practices and Web services Profiles*, Developerworks
- *IBM WebSphere V5.0 Performance, Scalability, and High Availability*, SG24-6198
- *Patterns: Direct Connections for Intra- and Inter-enterprise*, SG24-6933
- The SOAP 1.1 specification can be downloaded from: [http://www.w3.org/TR/SOAP/](http://www.w3.org/TR/SOAP/)
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.
Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

- server™
- server™

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.